CLASES: PROYECTO FINAL

David Agudelo Ochoa

Erika Dayana León Quiroga

Universidad de Antioquia
Despartamento de Ingeniería Electrónica y Telecomunicaciones
Informática II
Medellín-Antioquia
Abril de 2021

Índice

1. Sección introductoria.

En este informe se realizará la descripción de las clases que se implementarán para el desarrollo del proyecto final de la materia Informática II, esto nos permitirá planear el desarrollo de nuestro proyecto de manera que ya se tenga una idea general de este a la hora de iniciar con la codificación de nuestras clases. También nos ayudará a organizar mejor las ideas que tengamos para nuestro juego y estructurarlo de la mejor manera posible.

2. Idea de juego.

El juego tratará de una madre lobo (nuestra heroína) quien intentará salvar a sus cachorros de la amenaza inminente del ser humano, el cual quiere apropiarse de los poderes sobrenaturales que los cachorros heredaron de su madre. Para esto iniciará una persecución en donde se enfrentará a los secuestradores de sus crías. Tendremos dos niveles de dificultad, en el primero nuestra heroína tendrá que esquivar proyectiles lanzados de manera rectilínea hacia ella, los cuales cambiarán su velocidad y frecuencia de aparición a medida que el tiempo avance, aumentando así la dificultad del mismo. Una vez superada esta primera etapa, el tipo de lanzamiento del proyectil cambiará, y se usará para este MOVIMIENTO PARABÓLICO y un SISTEMA FÍSICO ELÁSTICO (resorte, energía mecánica) el cual dará la velocidad inicial de los proyectiles en este nivel, la dificultad aumentará a medida que avance el nivel, cambiando la frecuencia de los proyectiles y su alcance máximo. El último sistema físico requerido, se utilizará para la ambientación del juego y será MOVIMIENTO ARMÓNICO SIMPLE (un péndulo) en un reloj que muestre el tiempo restante del nivel actual.

3. Clases.

Para la creación de los mapas mentales que nos ayudarán con la estructuración de las clases, se utilizará el siguiente código de color: Verde para las clases, amarillo para las clases abstractas, blanco para los atributos, azul para los métodos y morado para los métodos abstractos.

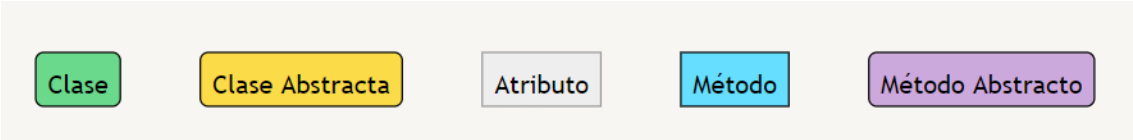

Figura 1: Código de color para los mapas mentales.

3.1. ObjetoAnimado.

- **Atributos:** En esta clase Objeto animado, tendremos como atributos la posición (x,y), el tamaño y número de frames el cual es necesario para realizar la animación de los objetos.
- **Métodos:** En los métodos tenemos Animar, en donde se actualizarán los frames de los objetos que queramos animar cada cierto tiempo, para esto se necesitarán dos parámetros, el número de frames de la animación y el path de cada una de las imágenes para crear la animación completa, para obtener este último parámetro se utilizará el retorno del método stringPath el cual nos ayudará a contruir el nombre de la dirección del archivo dependinedo del número de frame.

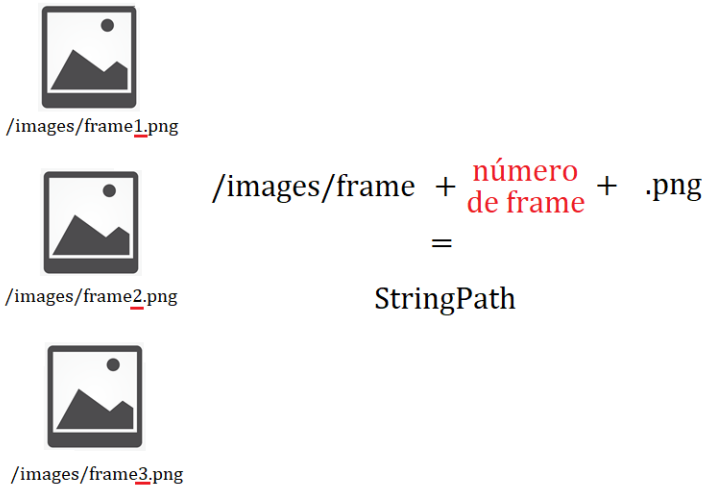

Figura 2: Ejemplo del funcionamiento del método stringPath. Como podemos ver, la primera y última parte del string se mantiene constante mientras que el número de frame va cambiando para lograr la animación de los objetos presentes en la escena.

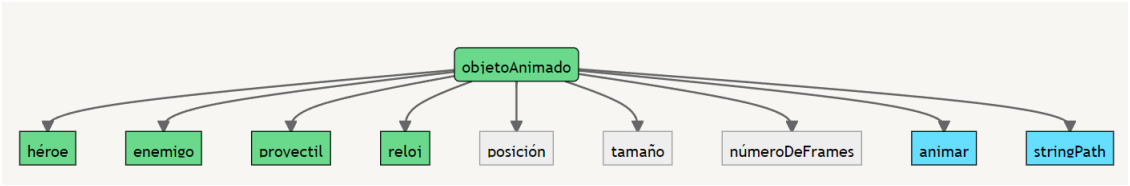

Figura 3: Mapa mental de la clase objetoAnimado.

3.1.1. Héroe

- **Atributos:** Dentro los atributos de la clase Héroe encontraremos el número de vidas, si el héroe dispone de su poder en un nivel determinado y por último el nivel en el que se encuentra.
- **Métodos:** Tendremos dos métodos, la actualización de la posición del héroe en Y cuando este realice un salto, y la verificación de la colisión del héroe con los límites del escenario.

3.1.2. Enemigo

- **Atributos:** Esta clase tendrá un único atributo que cambiará el nivel de dificultad dentro de un mismo nivel a medida que avance el tiempo.

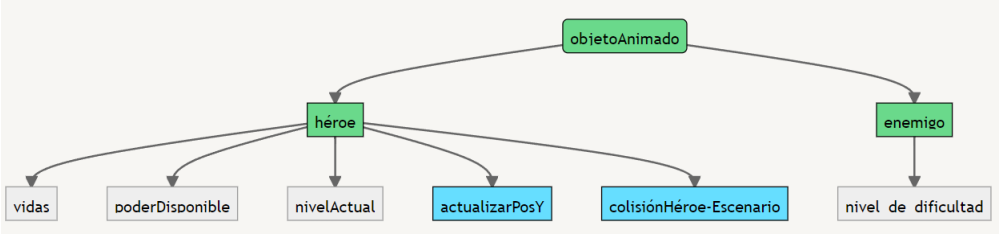

Figura 4: Mapa mental de la clase héroe y de la clase enemigo.

3.1.3. Proyectil

- **Atributos:** En este caso tendremos de atributos la velocidad inicial del proyectil y la frecuencia de aparición de este.
- **Métodos:** Dentro de los métodos encontraremos la verificación de la colisión entre el proyectil y el héroe, también la colisión entre el proyectil y los límites del escenario, la descripción del movimiento (MRU) usado para el proyectil en el nivel 1, y por último un método que describa el movimiento de tiro parabólico con resorte utilizado para el proyectil del nivel 2.

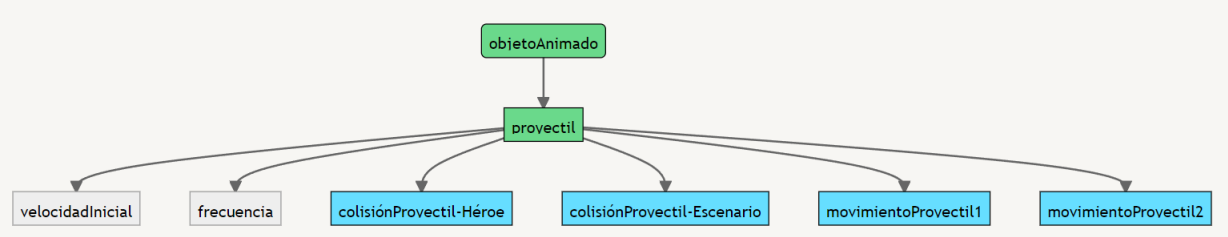

Figura 5: Mapa mental de la clase proyectil.

3.1.4. Reloj

- **Atributos:** Tendrá un atributo que nos permitirá llevar la cuenta del tiempo de la partida.
- **Métodos:** Tendremos un método que nos permitirá modelar el movimiento de un péndelo de reloj, el cual hará parte de la ambientación del juego. También se necesitará otro método para actualizar la visualización del tiempo de juego.

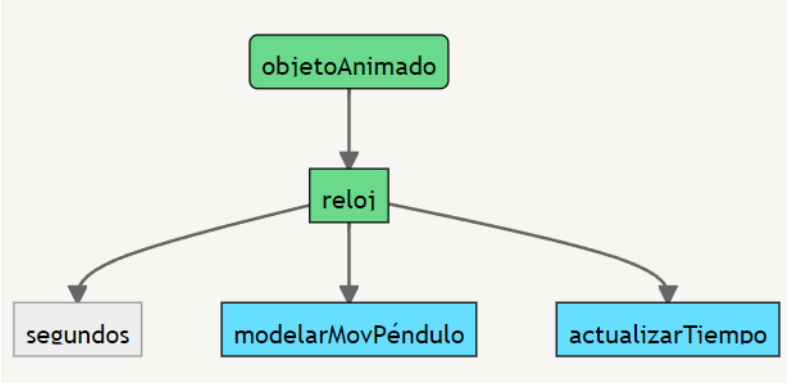

Figura 6: Mapa mental de la clase reloj.

3.2. Nivel.

Para la clase nivel se decidió que fuese de tipo abstracto, pues posee métodos que cambiarán dependiendo de las clases que los hereden.

- **Atributos:** La clase Niveles tendrá como único atributo el timepo de duración del nivel.
- **Métodos:** Tendremos el método para guardar el nivel, el cual nos ayudará a almacenar datos importantes del nivel actual, como la posición del héroe, las vidas de este, el tiempo restante de partida, nivel de dificultad y nivel en el que se encuentra.

- **Métodos abstractos:** Con respecto a los métodos abstractos, tendremos uno que nos permitirá iniciar el nivel correspondiente desde cero, creando los objetos para el funcionamiento del mismo y por último, el método para cargar el nivel guardado, el cual se encargará de tomar los datos guardados y recrear el nivel partiendo de estos.

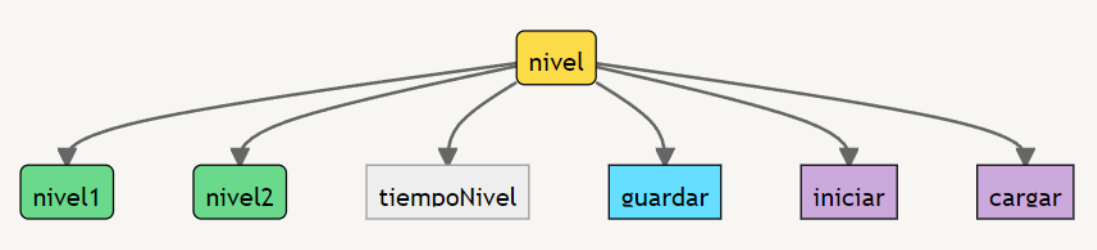

Figura 7: Mapa mental de la clase niveles.

3.2.1. Nivel-1

- **Métodos:** El método iniciar instanciará los objetos necesarios para la creación del nivel 1, dependiendo de las características de este. Para el método cargar se utilizarán los datos almacenados con ayuda del método guardar y de esta forma retomar el nivel en el punto guardado.

3.2.2. Nivel-2

- **Métodos:** El método inciar instanciará los objetos necesarios para la creación del nivel 2, dependiendo de las características de este. Para el método cargar, se utilizarán los datos almacenados con ayuda del método guardar y de esta forma retomar el nivel en el punto guardado.

4. Conclusiones.

- El proceso de estructurar las clases sin desarrollar código es bastante complejo, pues todo parte mucho de la especulación, ya que no hay garantías de si lo que se planeó podrá implementarse de manera exitosa.
- A pesar del comentario del ítem anterior, una vez alcanzamos un buen porcentaje de clases definidas, nos dimos cuenta de la simplicidad que adquiere la estructuración, pues nos garantiza que más adelante, una vez se empiece a codificar, exista una buena organización de las clases y una correcta descripción de sus métodos y atributos.
- El trabajo en equipo hace más ameno el desarrollo de los proyectos, y a pesar de que a veces puedan haber desacuerdos, mientras haya buena comunicación, no hay problema que no pueda solucionarse.